

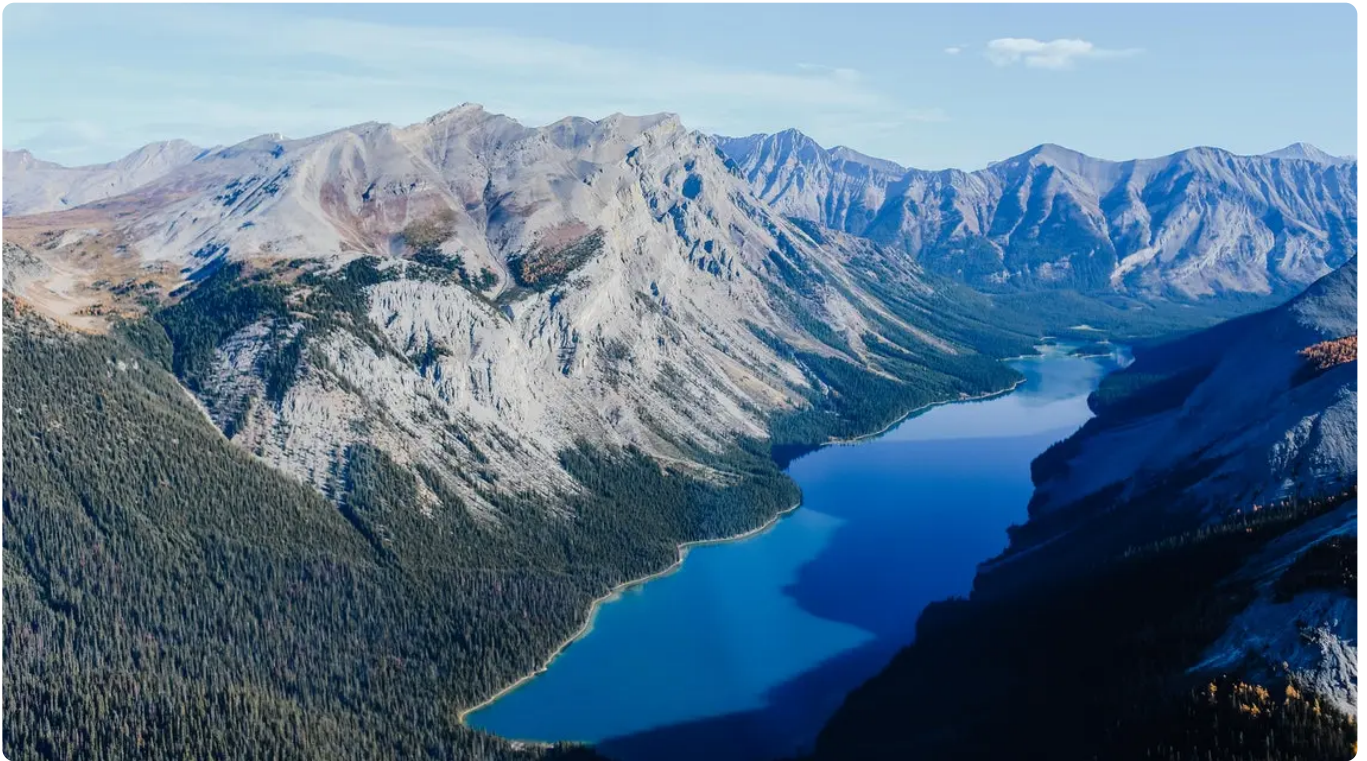


06 | switch表达式：怎么简化多情景操作？

2021-11-26 范学雷

《深入剖析Java新特性》

[课程介绍 >](#)



讲述：范学雷

时长 17:46 大小 16.28M



你好，我是范学雷。今天，我们聊一聊 Switch 表达式。

switch 表达式这个特性，首先在 JDK 12 中以预览版的形式发布。在 JDK 13 中，改进的 switch 表达式再次以预览版的形式发布。最后，switch 表达式在 JDK 14 正式发布。

不论你学习什么样的编程语言，合理地分析、判断、处理不同的情况都是必备的基本功。比如我们使用的 if-else 语句，还有 switch 语句，都是用来处理种种不同的情况的。我们都知道 switch 语句，那么 switch 表达式又是什么呢？switch 语句和 switch 表达式又有什么不同呢？



如果你了解了 Java 的语句和表达式这两个基本概念，你的困扰也许会少一点。Java 规范里，表达式完成对数据的操作。一个表达式的结果可以是一个数值 ($i * 4$)；或者是一个

变量（`i = 4`）；或者什么都不是（`void` 类型）。

Java 语句是 Java 最基本的可执行单位，它本身不是一个数值，也不是一个变量。Java 语句的标志性符号是分号（代码）和双引号（代码块），比如 `if-else` 语句，赋值语句等。这样再来看，就很简单了：`switch` 表达式就是一个表达式，而 `switch` 语句就是一个语句。

`switch` 表达式是什么样子的？为什么需要 `switch` 表达式？我们还是通过案例和代码，一点一点地来学习 `switch` 表达式吧。

阅读案例

在讲解或者学习 `switch` 语句时，每年的十二个月或者每周的七天，是我们经常使用的演示数据。在这个案例里，我们也使用这样的数据，来看看传统的 `switch` 语句有哪些需要改进的地方。

下面，我们要讨论的，也是一个传统的问题：该怎么用代码计算一个月有多少天？生活中，我们熟悉这样的顺口溜，“一三五七八十腊，三十一天永不差，四六九冬三十整，平年二月二十八，闰年二月把一加”。

下面的这段代码，就是按照这个顺口溜的逻辑来计算了一下，今天所在的这个月，一共有多少天。

 复制代码

```
1 package co.ivj.jus.swexpr.former;
2
3 import java.util.Calendar;
4
5 class DaysInMonth {
6     public static void main(String[] args) {
7         Calendar today = Calendar.getInstance();
8         int month = today.get(Calendar.MONTH);
9         int year = today.get(Calendar.YEAR);
10
11         int daysInMonth;
12         switch (month) {
13             case Calendar.JANUARY:
14             case Calendar.MARCH:
15             case Calendar.MAY:
16             case Calendar.JULY:
17             case Calendar.AUGUST:
18             case Calendar.OCTOBER:
```

```
19         case Calendar.DECEMBER:
20             daysInMonth = 31;
21             break;
22         case Calendar.APRIL:
23         case Calendar.JUNE:
24         case Calendar.SEPTEMBER:
25         case Calendar.NOVEMBER:
26             daysInMonth = 30;
27             break;
28         case Calendar.FEBRUARY:
29             if (((year % 4 == 0) && !(year % 100 == 0))
30                 || (year % 400 == 0)) {
31                 daysInMonth = 29;
32             } else {
33                 daysInMonth = 28;
34             }
35             break;
36         default:
37             throw new RuntimeException(
38                 "Calendar in JDK does not work");
39     }
40
41     System.out.println(
42         "There are " + daysInMonth + " days in this month.");
43 }
44 }
```

这段代码里，我们使用了 switch 语句。代码本身并没有什么错误，但是，至少有两个容易犯错误的地方。

第一个容易犯错的地方，就是在 break 关键字的使用上。上面的代码里，如果多使用一个 break 关键字，代码的逻辑就会发生变化；同样的，少使用一个 break 关键字也会出现问题。

[复制代码](#)

```
1 int daysInMonth;
2 switch (month) {
3     case Calendar.JANUARY:
4     case Calendar.MARCH:
5     case Calendar.MAY:
6         break;    // WRONG BREAK!!!
7     case Calendar.JULY:
8     case Calendar.AUGUST:
9     case Calendar.OCTOBER:
10    case Calendar.DECEMBER:
11        daysInMonth = 31;
```

```
12         break;
13     // snipped
14 }
```

 复制代码

```
1  int daysInMonth;
2  switch (month) {
3      // snipped
4      case Calendar.APRIL:
5      case Calendar.JUNE:
6      case Calendar.SEPTEMBER:
7      case Calendar.NOVEMBER:
8          daysInMonth = 30;
9          // WRONG, NO BREAK!!!
10     case Calendar.FEBRUARY:
11         if (((year % 4 == 0) && !(year % 100 == 0))
12             || (year % 400 == 0)) {
13             daysInMonth = 29;
14         } else {
15             daysInMonth = 28;
16         }
17         break;
18     // snipped
19 }
```

break 语句的遗漏或者冗余，这样的错误如此得常见，甚至于被单列成了一个 [🔗 常见软件安全漏洞](#)。凡是使用 switch 语句的代码，都有可能成为黑客们重点关注的对象。由于逻辑的错误和黑客的特殊关照，我们在编写代码的时候，需要十二分的小心；阅读代码的时候，也需要反复地查验 break 语句的前后语境。毫无疑问，这增加了代码维护的成本，降低了生产效率。

为什么 switch 语句里需要使用 break 呢？最主要的原因，就是希望能够在不同的情况下，共享部分或者全部的代码片段。比如上面的例子中，四月、六月、九月、十一月这四种情景，可以共享每个月都是 30 天这样的代码片段。这个代码片段只需要写在十一月情景的后面，前面的四月、六月和九月这三个情景都会顺次执行下面的操作（fall-through），直到遇到下一个 break 语句或者 switch 语句终结。

现在我们都知道了，这样是一个弊大于利的设计。但很遗憾，Java 初始的设计就是采用了这样的设计思想。如果要新设计一门现代的语言，我们需要更多地使用 switch 语句，但是

就不要再使用 `break` 语句了。不过，不同的情景共享代码片段，仍然是一个真实的需求。在废弃掉 `break` 语句之前，我们要找到在不同的情景间共享代码片段的新规则。

第二个容易犯错的地方，是反复出现的赋值语句。在上面的代码中，**`daysInMonth`** 这个本地变量的变量声明和实际赋值是分开的。赋值语句需要反复出现，以适应不同的情景。如果在 `switch` 语句里，**`daysInMonth`** 变量没有被赋值，编译器也不会报错，缺省的或者初始的变量值就会被使用。

 复制代码

```
1  int daysInMonth = 0;
2  switch (month) {
3      // snipped
4      case Calendar.APRIL:
5      case Calendar.JUNE:
6      case Calendar.SEPTEMBER:
7      case Calendar.NOVEMBER:
8          break;    // WRONG, INITIAL daysInMonth value IS USED!!!
9      case Calendar.FEBRUARY:
10         // snipped
11     }
```

在上面的例子里，初始的变量值不是一个合适的数；当然，在另外一个例子里，缺省的或者初始的变量值也可能就是一个合适的数了。为了判断这个本地变量有没有合适的值，我们需要通览整个 `switch` 语句块，确保赋值没有遗漏，也没有多余。这增加了编码出错的几率，也增加了阅读代码的成本。

那么，能不能让多情景处理的代码块拥有一个数值呢？或者换个说法，多情景处理的代码块能不能变成一个表达式？这个想法，就催生了 Java 语言的新特性：“`switch` 表达式”。

switch 表达式

`switch` 表达式是什么样子的呢？下面的这段代码，使用的就是 `switch` 表达式，它改进了上面阅读案例里的代码。你可以带着上面遇到的问题，来阅读这段代码。这些问题包括：

switch 表达式是怎么表示一个数值，从而可以给变量赋值的？

在不同的情景间，switch 表达式是怎么共享代码片段的？

使用 switch 表达式的代码，有没有变得更简单、更皮实、更容易理解？

 复制代码

```
1 package co.ivj.jus.swexpr.modern;
2
3 import java.util.Calendar;
4
5 class DaysInMonth {
6     public static void main(String[] args) {
7         Calendar today = Calendar.getInstance();
8         int month = today.get(Calendar.MONTH);
9         int year = today.get(Calendar.YEAR);
10
11         int daysInMonth = switch (month) {
12             case Calendar.JANUARY,
13                  Calendar.MARCH,
14                  Calendar.MAY,
15                  Calendar.JULY,
16                  Calendar.AUGUST,
17                  Calendar.OCTOBER,
18                  Calendar.DECEMBER -> 31;
19             case Calendar.APRIL,
20                  Calendar.JUNE,
21                  Calendar.SEPTEMBER,
22                  Calendar.NOVEMBER -> 30;
23             case Calendar.FEBRUARY -> {
24                 if (((year % 4 == 0) && !(year % 100 == 0))
25                     || (year % 400 == 0)) {
26                     yield 29;
27                 } else {
28                     yield 28;
29                 }
30             }
31             default -> throw new RuntimeException(
32                 "Calendar in JDK does not work");
33         };
34
35         System.out.println(
36             "There are " + daysInMonth + " days in this month.");
37     }
38 }
```

我们最先看到的变化，就是 switch 代码块出现在了赋值运算符的右侧。这也就意味着，这个 switch 代码块表示的是一个数值，或者是一个变量。换句话说，这个 switch 代码块是一个表达式。

 复制代码

```
1 int daysInMonth = switch (month) {  
2     // snipped  
3 }
```

我们看到的第二个变化，是多情景的合并。也就是说，一个 case 语句，可以处理多个情景。这些情景，使用逗号分隔开来，共享一个代码块。而传统的 switch 代码，一个 case 语句只能处理一种情景。

 复制代码

```
1 case Calendar.JANUARY,  
2     Calendar.MARCH,  
3     // snipped
```

多情景的合并的设计，满足了共享代码片段的需求。而且，由于只使用一个 case 语句，也就不再需要使用 break 语句来满足这个需求了。所以，break 语句从 switch 表达式里消失了。

不同之处在于，传统的 switch 代码，不同的 case 语句之间可以共享部分的代码片段；而 switch 表达式里，需要共享全部的代码片段。这看似是一个损失，但其实，共享部分代码片段的能力给代码的编写者带来的困惑远远多于它带来的好处。如果需要共享部分的代码片段，我们总是可以找到替换的办法，比如把需要共享的代码封装成更小的方法。所以，我们没有必要担心 switch 表达式不支持共享部分代码片段。

下一个变化，是一个新的情景操作符，“->”，它是一个箭头标识符。这个符号使用在 case 语句里，一般化的形式是“case L ->”。这里的 L，就是要匹配的一个或者多个情景。如果目标变量和情景匹配，那么就执行操作符右边的表达式或者代码块。如果要匹配的情景有两个或者两个以上，就要使用逗号“,”用分隔符把它们分割开来。

 复制代码

```
1 case Calendar.JANUARY,  
2     // snipped  
3     Calendar.DECEMBER -> 31;
```

传统的 switch 代码，这个一般化的形式是 “case L :”，也就是使用冒号标识符。为什么不延续使用传统的情景操作符呢？这主要是出于简化代码的考虑。我们依然可以在 switch 表达式里使用冒号标识符，使用冒号标识符的一个 case 语句只能匹配一个情景，这种情况我们稍后再讨论。

下一个我们看到的變化，是箭头标识符右侧的数值。这个数值，代表的就是该匹配情景下，switch 表达式的数值。需要注意的是，箭头标识符右侧可以是表达式、代码块或者异常抛出语句，而不能是其他的形式。如果只需要一个语句，这个语句也要以代码块的形式呈现出来。

 复制代码

```
1 case Calendar.JANUARY,  
2     // snipped  
3     Calendar.DECEMBER -> { // CORRECT, enclosed with braces.  
4     yield 31;  
5 }
```

没有以代码块形式呈现的代码，编译的时候，就会报错。这是一个很棒的约束。代码块的形式，增强了视觉效果，减少了编码的失误。在 [🔗 《代码精进之路》](#) 这个专栏里，我们反复强调过这种形式的好处。

 复制代码

```
1 case Calendar.JANUARY,  
2     // snipped  
3     Calendar.DECEMBER -> // WRONG, not a block.  
4     yield 31;
```

另外，箭头标识符右侧需要一个表达 switch 表达式的数值，这是一个很强的约束。如果一个语句破坏了这个需要，它就不能出现在 switch 表达式里。比如，下面的代码里的 return 语句，意图退出该方法，而没有表达这个 switch 表达式的数值。这段代码就不能通过编译器的审查。

 复制代码

```
1 int daysInMonth = switch (month) {  
2     // snipped  
3     case Calendar.APRIL,  
4         // snipped
```

```
5         Calendar.NOVEMBER -> {
6             // yield 30;
7             return; // WRONG, return outside of enclosing switch expression.
8         }
9         // snipped
10    }
```

最后一个我们能够看到的变化，是出现了一个新的关键字“yield”。大多数情况下，switch 表达式箭头标识符的右侧是一个数值或者是一个表达式。如果需要一个或者多个语句，我们就要使用代码块的形式。这时候，我们就需要引入一个新的 yield 语句来产生一个值，这个值就成为这个封闭代码块代表的数值。

为了便于理解，我们可以把 yield 语句产生的值看成是 switch 表达式的返回值。所以，yield 只能用在 switch 表达式里，而不能用在 switch 语句里。

[复制代码](#)

```
1  case Calendar.FEBRUARY -> {
2      if (((year % 4 == 0) && !(year % 100 == 0))
3          || (year % 400 == 0)) {
4          yield 29;
5      } else {
6          yield 28;
7      }
8  }
```

其实，这里还有一个我们从上述的代码里看不到的变化。在 switch 表达式里，所有的情景都要列举出来，不能多、也不能少（这也就是我们常说的穷举）。

比如说，在上面的例子里，如果没有最后的 default 情景分支，编译器就会报错。这是一个影响深远的改进，它会使得 switch 表达式的代码更加健壮，大幅度降低维护成本，如果未来需要增加一个情景分支的话，就更是如此了。

[复制代码](#)

```
1  int daysInMonth = switch (month) {
2      case Calendar.JANUARY,
3          // snipped
4          Calendar.DECEMBER -> 31;
5      case Calendar.APRIL,
6          // snipped
7          Calendar.NOVEMBER -> 30;
```

```
8     case Calendar.FEBRUARY -> {
9         // snipped
10    }
11    // WRONG to comment out the default branch, 'switch' expression
12    // MUST cover all possible input values.
13    //
14    // default -> throw new RuntimeException(
15    //         "Calendar in JDK does not work");
16 };
```

改进的 switch 语句

通过上面的解读, 我们知道了 switch 表达式里有很多积极的变化。那这些变化有没有影响 switch 语句呢? 比如说, 我们能够在 switch 语句里使用箭头标识符吗? 我们前面说过, yield 语句是用来产生一个 switch 表达式代表的数值的, 因此 yield 语句只能用在 switch 表达式里, 不能用在 switch 语句。

其他的变化呢? 我们还是先来看下面一段代码。

[复制代码](#)

```
1 private static int daysInMonth(int year, int month) {
2     int daysInMonth = 0;
3     switch (month) {
4         case Calendar.JANUARY,
5             Calendar.MARCH,
6             Calendar.MAY,
7             Calendar.JULY,
8             Calendar.AUGUST,
9             Calendar.OCTOBER,
10            Calendar.DECEMBER ->
11            daysInMonth = 31;
12        case Calendar.APRIL,
13            Calendar.JUNE,
14            Calendar.SEPTEMBER,
15            Calendar.NOVEMBER ->
16            daysInMonth = 30;
17        case Calendar.FEBRUARY -> {
18            if (((year % 4 == 0) && !(year % 100 == 0))
19                || (year % 400 == 0)) {
20                daysInMonth = 29;
21                break;
22            }
23
24            daysInMonth = 28;
25        }
26        // default -> throw new RuntimeException(
```

```
27         // "Calendar in JDK does not work");
28     }
29
30     return daysInMonth;
31 }
```

在这段代码里，我们看到了箭头标识符，看到了 break 语句，看到了注释掉的 default 语句。这是一段合法的、能够工作的代码。换个说法，switch 语句可以使用箭头标识符，也可以使用 break 语句，也不需要列出所有的情景。表面上看起来，switch 语句的改进不是那么显而易见。其实，switch 语句的改进主要体现在 break 语句的使用上。

我们应该也看到了，break 语句没有出现在下一个 case 语句之前。这也就意味着，使用箭头标识符的 switch 语句不再需要 break 语句来实现情景间的代码共享了。虽然我们还可以通过这样使用 break 语句，但是已经不再必要了。

 复制代码

```
1 switch (month) {
2     // snipped
3     case Calendar.APRIL,
4         // snipped
5         Calendar.NOVEMBER -> {
6         daysInMonth = 30;
7         break; // UNNECESSARY, could be removed safely.
8     }
9     // snipped
10 }
```

有没有 break 语句，使用箭头标识符的 switch 语句都不会顺次执行下面的操作（fall-through）。这样，我们前面谈到的 break 语句带来的烦恼也就消失不见了。

不过，使用箭头标识符的 switch 语句并没有禁止 break 语句，而是恢复了它本来的意义：从代码片段里抽身，就像它在循环语句里扮演的角色一样。

 复制代码

```
1 switch (month) {
2     // snipped
3     case Calendar.FEBRUARY -> {
4         if (((year % 4 == 0) && !(year % 100 == 0))
5             || (year % 400 == 0)) {
6             daysInMonth = 29;
7         }
8     }
9     // snipped
10 }
```

```
7         break;        // BREAK the switch statement
8     }
9
10    daysInMonth = 28;
11 }
12 // snipped
13 }
```

怪味的 switch 表达式

我们前面说过，switch 表达式也可以使用冒号标识符。使用冒号标识符的一个 case 语句只能匹配一个情景，而且支持 fall-through。和箭头标识符的 switch 表达式一样，使用冒号标识符 switch 表达式也不支持 break 语句，取而代之的是 yield 语句。

这是一个充满了怪味道的编码形式，我并不推荐使用这种形式，但我可以带你略作了解。下面的这段代码，就是我们试着把箭头标识符替换成冒号标识符的一个例子。你可以比较一下使用冒号标识符和箭头标识符的两段代码，想一想两种不同形式的优劣。毫无疑问，使用箭头标识符的代码更加简洁。

[📄 复制代码](#)

```
1 package co.ivi.jus.swexpr.legacy;
2
3 import java.util.Calendar;
4
5 class DaysInMonth {
6     public static void main(String[] args) {
7         Calendar today = Calendar.getInstance();
8         int month = today.get(Calendar.MONTH);
9         int year = today.get(Calendar.YEAR);
10
11         int daysInMonth = switch (month) {
12             case Calendar.JANUARY:
13             case Calendar.MARCH:
14             case Calendar.MAY:
15             case Calendar.JULY:
16             case Calendar.AUGUST:
17             case Calendar.OCTOBER:
18             case Calendar.DECEMBER:
19                 yield 31;
20             case Calendar.APRIL:
21             case Calendar.JUNE:
22             case Calendar.SEPTEMBER:
23             case Calendar.NOVEMBER:
24                 yield 30;
25             case Calendar.FEBRUARY:
```

```
26         if (((year % 4 == 0) && !(year % 100 == 0))
27             || (year % 400 == 0)) {
28             yield 29;
29         } else {
30             yield 28;
31         }
32     default:
33         throw new RuntimeException(
34             "Calendar in JDK does not work");
35     };
36
37     System.out.println(
38         "There are " + daysInMonth + " days in this month.");
39 }
40 }
```

有了使用箭头标识符的 switch 语句和 switch 表达式之后，我们不再推荐使用冒号标识符的 switch 语句和 switch 表达式。学习并使用箭头标识符的 switch 语句和 switch 表达式，会使代码更简洁、更健壮。

总结

好，到这里，我来做个小结。从前面的讨论中，我们重点了解了 switch 表达式和改进的 switch 语句。我们还讨论了 switch 表达式带来的新概念和新的关键字，了解了这些基本概念以及它们的适用范围。

新的 switch 形式、语句和表达式，不同的使用范围，这些概念交织在一起，让 switch 的学习和使用都变成了一件有点挑战性的事情。箭头标识符的引入，简化了代码，提高了编码效率。可是，学习这么多种 switch 的表现形式，也增加了我们的学习负担。为了帮助你快速掌握这些形式，我把不同的 switch 表达形式，以及它们支持的特征，放在了下面这张表格里。

	使用箭头标识符 的switch表达式	使用箭头标识符 的switch语句	使用冒号标识符 的switch表达式	使用冒号标识符 的switch语句
break语句	✗	✓	✗	✓
yield语句	✓	✗	✓	✗
情景穷举	✓	✗	✓	✗
fall-through	✗	✗	✓	✓
多情景匹配	✓	✓	✗	✗



或者，你也可以记住下面的总结：

- break 语句只能出现在 switch 语句里，不能出现在 switch 表达式里；
- yield 语句只能出现在 switch 表达式里，不能出现在 switch 语句里；
- switch 表达式需要穷举出所有的情景，而 switch 语句不需要情景穷举；
- 使用冒号标识符的 swtich 形式，支持情景间的 fall-through；而使用箭头标识符的 switch 形式不支持 fall-through；
- 使用箭头标识符的 swtich 形式，一个 case 语句支持多个情景；而使用冒号标识符的 switch 形式不支持多情景的 case 语句。

使用箭头标识符的 swtich 形式，废止了容易出问题的 fall-through 这个特征。因此，我们推荐使用箭头标识符的 swtich 形式，逐步废止使用冒号标识符的 swtich 形式。在 switch 表达式和 switch 语句之间，我们应该优先使用 switch 表达式。这些选择，都可以帮助我们简化代码逻辑，减少代码错误，提高生产效率。

如果你要丰富你的代码评审清单，学习完这一节内容后，你可以加入下面这一条：

- 使用冒号标识符的 swtich 形式，是不是可以更改为使用箭头标识符？
- 使用 switch 语句赋值的操作，是不是可以更改为使用 switch 表达式？

另外，我还拎出了几个今天讨论过的技术要点，这些都可能在你面试中出现哦。通过这次学习，你应该能够：

知道 switch 表达式，并且能够使用 switch 表达式；

- 面试问题：你知道 switch 表达式吗？该怎么处理 switch 表达式里的语句？

了解 switch 表达式要解决的问题，并且知道解决掉这些问题的办法；

- 面试问题：使用 switch 表达式有哪些好处？

了解不同的 switch 的表现形式，能够看得懂不同的表现形式，并且给出改进意见。

- 面试问题：你更喜欢使用箭头标识符还是冒号标识符？

如果你能够有意识地使用箭头标识符的 switch 表达式，应该可以大幅度提高编码的效率和质量；如果你能够了解不同的 switch 表现形式，并且对每种形式都有自己的见解，你就能帮助你的同事提高编码的效率和质量。毫无疑问，在面试的时候，有意识地在代码里使用 switch 表达式，是一个能够展现你的学习能力、理解能力和对新知识的接受能力的一个好机会。

思考题

在前面的讨论里，我们说过情景穷举是一个影响深远的改进方向，它会使得 switch 表达式的代码更加健壮，大幅度降低维护成本，特别是在未来需要增加一个情景分支的情形下。但是，限于篇幅，我们并没有详细地展开讨论其中的细节。现在，我们把这个讨论当作一个稍微有点挑战的思考题。

假设有一天，地球和太阳的关系发生了变化，这种变化还没有大到毁灭人类的程度，但是也足以改变年月的关系了。于是，天文学家重新修订了日历，增加了一个新的月份，第十三个月。为了对应这种变化，JDK 的设计者们也给 Calendar 类增加了第十三个月：Calendar.AFTERDEC。那么，我们的问题就来了。

第一个问题是，我们现在的代码能够检测到这个变化吗？如果不能，是不是只有系统崩溃的时候，我们才能够意识到问题的存在？

第二个问题是，有没有更健壮的设计，能够帮助我们在系统崩溃之前就能够检测到这个意想不到的变化？从而给我们留出时间更改我们的代码和系统？

稍微提示一个，解决这个问题其中一个思路，就是要使用有穷举能力的表达式，然后设计出可以表达穷举情景的新形式，而不是使用泛泛的整数来表达十二个月。

我在下面的例子中写了一个代码小样。这个代码小样，实现的还是一年只有十二个月的逻辑。现在我们假设，一年还是十二个月，但是我们想让这段代码健壮到能够检测到未来一年变成十一个月或者十三个月的情景。

在这个代码小样里，我也试着加入了一些提示。当然，你也可以试着找找其他的解决方案。请试着将这段代码修改成你喜欢的样子，让我们一起来看看怎么解决掉这个问题。

 复制代码

```
1 package co.ivj.jus.swexpr.review.xuelel;
2
3 import java.util.Calendar;
4
5 class DaysInMonth {
6     public static void main(String[] args) {
7         Calendar today = Calendar.getInstance();
8         int month = today.get(Calendar.MONTH);
9         int year = today.get(Calendar.YEAR);
10
11         // Hints: could we replace the integer month
12         // with an exhaustive enumeration?
13         int daysInMonth = switch (month) {
14             case Calendar.JANUARY,
15                  Calendar.MARCH,
16                  Calendar.MAY,
17                  Calendar.JULY,
18                  Calendar.AUGUST,
19                  Calendar.OCTOBER,
20                  Calendar.DECEMBER -> 31;
21             case Calendar.APRIL,
22                  Calendar.JUNE,
23                  Calendar.SEPTEMBER,
24                  Calendar.NOVEMBER -> 30;
25             case Calendar.FEBRUARY -> {
26                 if (((year % 4 == 0) && !(year % 100 == 0))
27                     || (year % 400 == 0)) {
28                     yield 29;
29                 } else {
30                     yield 28;
31                 }
32             }
33             // Hints: Are we able to replace the default case by
34             // enumerating all cases with case clause above?
35             default -> throw new RuntimeException(
```

```
36         "Calendar in JDK does not work");
37     };
38
39     System.out.println(
40         "There are " + daysInMonth + " days in this month.");
41 }
42 }
```

欢迎你在留言区留言、讨论，分享你的阅读体验以及你对这个思考题的想法。

注：本文使用的完整的代码可以从 [🔗 GitHub](#) 下载，你可以通过修改 [🔗 GitHub](#) 上 [🔗 review template](#) 代码，完成这次的思考题。如果你想要分享你的修改或者想听听评审的意见，请提交一个 GitHub 的拉取请求（Pull Request），并把拉取请求的地址贴到留言里。这一小节的拉取请求代码，请在 [🔗 实例匹配专用的代码评审目录](#) 下，建一个以你的名字命名的子目录，代码放到你专用的子目录里。比如，我的代码，就放在 `swexpr/review/xuelel` 的目录下面。

分享给需要的人，Ta 订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 3

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 类型匹配：怎么切除臃肿的强制转换？

下一篇 07 | switch 匹配：能不能适配不同的类型？

精选留言 (8)

 写留言



ABC

2021-11-26

第一个问题是：我们现在的代码能够检测到这个变化吗？如果不能，是不是只有系统崩溃的时候，我们才能够意识到问题的存在？

答：如果是新增的月份现在的设计只能抛出异常,提醒我们修改代码.如果是减少了月份,在未更新JDK的情况下,代码正常运行.另外: JDK并非强更新,有些用户并不会频繁更新JDK,在...

展开 ∨

作者回复: “就三行代码，提pr有点太寒酸了”，多小的改动都不会显得寒酸。任何一个开源社区，都需要很多小的改动。

共 5 条评论 >  2



aoe

2021-11-26

->好用很多，switch 变好看了

展开 ∨

作者回复: 好看就是生产力！

  2



fatme

2021-12-02

终于明白思考题的思路，原来是要在编译期暴露出问题。这种思路，即使是对 jdk 8 以下也是有用的，只不过是像 switch 表达式一样有语言本身的帮助。又学了一招，谢谢老师！

作者回复: 不谢。

  2



猿人谷

2021-11-26

这个新特性看着很不错！

展开 ∨

作者回复: 它应该是我们以后使用switch的标准。

  2



xiaobang 

2021-11-27

第二个问题，感觉可以用枚举类型来表示月份，这样switch表达式检查到没有穷举到全部值

作者回复：是的，如果月份的API重新设计的话，一定是使用枚举而不是整数来表达。

**aoe**

2021-11-26

今天《极客时间》两个专栏同时更新，主题都是 switch

- 《深入剖析 Java 新特性》06 | Switch表达式：怎么简化多情景操作？
- 《Tony Bai · Go 语言第一课》20 | 控制结构：Go中的switch语句有哪些变化？

...

展开 ∨

作者回复：“综合能力 Go 的更灵活”，该怎么理解？从你给出的例子没有看出来。



共 6 条评论 >

**Jxin**

2021-11-26

- 1.既然知道是一个弊大于利的方案，为何 switch表达式 还要使用 冒号标识符 来保留fall-through的模式？switch表达式 区别于 switch语句，可没有兼容问题。
 - 2.猜测使用 -> 而不是：。应该是为了区别 switch语句：的语义，同时对称 Lambda 风格的表达方式。
 3. switch语句 本身好像直接被归为坏味道，不建议使用（eff里看到的还是哪，忘了）。...
- 展开 ∨

作者回复：1. 如果反对的声音再多一点，这个冒号的表达式形式可能就没了。

2. 需要和冒号的形式区别开来。

3. 能够支持有限的表达式啊。再看看下一讲吧，里边有相关的话题。

**大胖子呀、**

2021-11-26

第一个问题，目前的程序可以检测到多了一个月份，会通过最后的default语句抛出异常，但是检测不到少了一个月份；

第二个问题，我想着是否可以在default表达式里，通过遍历Calendar枚举的元素，去匹配出Calendar.AFTERDEC，但是我有疑问的是：月份多出了一个月，它的出现是否会导致已有月份的计算方式的改变？它的计算方式是按照31、30天，还是按照2月份，又或者是一...
展开 ∨

作者回复: 1. 多一个月检测到的方式是运行时抛出异常，这需要测试代码的配合，不能等到实际运行阶段。少一个月份，通常这个月份，比如FEBRUARY，就会从API里删除掉，这样编译时就可以检测出来了。

2. 你的疑问可以通过自己进一步的假设解决，你可以使用30*31*28，或者另外的一个你喜欢的数。

